



UNITED STATES PATENT AND TRADEMARK OFFICE

A

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/542,189	04/04/2000	Gordon Taylor Davis	RAL9-2000-0008-US1	5890
26675	7590	10/28/2005		
DRIGGS, LUCAS, BRUBAKER & HOGG CO. L.P.A. 38500 CHARDON ROAD DEPT. IRA WILLOUGBY HILLS, OH 44094			EXAMINER HUISMAN, DAVID J	
			ART UNIT	PAPER NUMBER
			2183	

DATE MAILED: 10/28/2005

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary	Application No. 09/542,189	Applicant(s) DAVIS ET AL.	
	Examiner David J. Huisman	Art Unit 2183	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

- 1) ☒ Responsive to communication(s) filed on 10 August 2005.
- 2a) ☐ This action is **FINAL**. 2b) ☒ This action is non-final.
- 3) ☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

- 4) ☒ Claim(s) 1,3,5-11,13,15-23,27 and 29-33 is/are pending in the application.
- 4a) Of the above claim(s) _____ is/are withdrawn from consideration.
- 5) ☐ Claim(s) _____ is/are allowed.
- 6) ☒ Claim(s) 1,3,5-11,13,15-23,27 and 29-33 is/are rejected.
- 7) ☐ Claim(s) _____ is/are objected to.
- 8) ☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

- 9) ☐ The specification is objected to by the Examiner.
- 10) ☒ The drawing(s) filed on 04 April 2000 is/are: a) ☒ accepted or b) ☐ objected to by the Examiner.
Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).
Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).
- 11) ☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

- 12) ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
- a) ☐ All b) ☐ Some * c) ☐ None of:
1. ☐ Certified copies of the priority documents have been received.
2. ☐ Certified copies of the priority documents have been received in Application No. _____.
3. ☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

- | | |
|---|---|
| 1) <input checked="" type="checkbox"/> Notice of References Cited (PTO-892) | 4) <input type="checkbox"/> Interview Summary (PTO-413) |
| 2) <input type="checkbox"/> Notice of Draftsperson's Patent Drawing Review (PTO-948) | Paper No(s)/Mail Date. _____ |
| 3) <input type="checkbox"/> Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08) | 5) <input type="checkbox"/> Notice of Informal Patent Application (PTO-152) |
| Paper No(s)/Mail Date _____ | 6) <input type="checkbox"/> Other: _____ |

DETAILED ACTION

1. Claims 1, 3, 5-11, 13, 15-23, 27, and 29-33 have been examined.

Papers Submitted

2. It is hereby acknowledged that the following papers have been received and placed of record in the file: RCE and Amendment as received on 8/10/2005.

Double Patenting

3. The nonstatutory double patenting rejection is based on a judicially created doctrine grounded in public policy (a policy reflected in the statute) so as to prevent the unjustified or improper timewise extension of the "right to exclude" granted by a patent and to prevent possible harassment by multiple assignees. See *In re Goodman*, 11 F.3d 1046, 29 USPQ2d 2010 (Fed. Cir. 1993); *In re Longi*, 759 F.2d 887, 225 USPQ 645 (Fed. Cir. 1985); *In re Van Ornum*, 686 F.2d 937, 214 USPQ 761 (CCPA 1982); *In re Vogel*, 422 F.2d 438, 164 USPQ 619 (CCPA 1970); and *In re Thorington*, 418 F.2d 528, 163 USPQ 644 (CCPA 1969).

A timely filed terminal disclaimer in compliance with 37 CFR 1.321(c) may be used to overcome an actual or provisional rejection based on a nonstatutory double patenting ground provided the conflicting application or patent is shown to be commonly owned with this application. See 37 CFR 1.130(b).

Effective January 1, 1994, a registered attorney or agent of record may sign a terminal disclaimer. A terminal disclaimer signed by the assignee must fully comply with 37 CFR 3.73(b).

4. Claims 1, 6, 8-10, 11, 16, and 18-20 are rejected under the judicially created doctrine of obviousness-type double patenting as being unpatentable over claims 2, 3, 4-6, 2, 3, and 4-6, respectively, of Davis et al., U.S. Patent No. 6,931,641 (herein referred to as Davis) in view of Kruse and Ryba, "Data Structures And Program Design in C++," 1999 (as applied in the previous Office Action and herein referred to as Kruse).

When comparing claim 1 of the instant application to claim 2 of Davis, it can be seen that the differences include:

Art Unit: 2183

a) claim 1 of the instant application refers to a “network processor” while claim 2 of Davis refers to a “processor.” However, the word “network” in claim 1 is recited in the preamble and does not breath life into the claim. Consequently, the word “network” is given no weight. However, even if “network” were given weight, the examiner asserts that network processors are well known in the art as being processors which efficiently handle applications such as network routing, packet processing, and any other network related functions. As a result, it would have been obvious to modify Davis’ processor to be a network processor in order to allow it to efficiently handle network applications.

b) claim 1 of the instant application refers to “accessible data available in a tree search structure” while claim 2 of Davis refers to “accessible data”. However, Kruse has taught that search trees are quick and efficient for inserting, deleting, and searching for data. See section 10.2 on pages 444-445 (supplied with a previous Office Action). As a result, it would have been obvious to modify Davis such that the data is accessible via a tree search structure.

c) claim 1 of the instant application refers to stalling “due to a short latency event” and stalling “due to a long latency event” while claim 2 of Davis refers to “a short latency event that causes execution of the first thread to stall for a first predefined time interval” and “a long latency event that causes execution of the first thread to stall for a second predefined time interval.” The examiner asserts that the differences here are inherent. That is, short and long latency events will inherently have first and second predefined time intervals associated with them (i.e., all stalling takes time).

d) claim 1 of the instant application refers to “retaining full control if the stall is due to a long latency event” while claim 2 of Davis refers to “control of the execution is not returned to the

Art Unit: 2183

first thread when execution of the first thread stalls due to a long latency event.” The examiner asserts that if there are two threads and control is not returned to the first thread, then the option would be to either have a next thread retain control or do nothing at all. As is known in the art, it is much more desirable to do some useful processing as opposed to the system sitting idle. Consequently, it would have been obvious to modify David such that a next thread retains control.

5. Claims 6 and 8-10 of the instant application are similarly rejected (as above) under the judicially created doctrine of obviousness-type double patenting as being unpatentable over claims 3, and 4-6 of Davis, respectively.

6. Claims 11, 16, and 18-20 of the instant application are similarly rejected (as above) under the judicially created doctrine of obviousness-type double patenting as being unpatentable over claims 2, 3, and 4-6 of Davis, respectively.

Claim Objections

7. Claims 13 and 15-23 recite either the limitation "The processing system" or "The system" line 1. There is insufficient antecedent basis for this limitation in the claim.

Claim Rejections - 35 USC § 103

8. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negated by the manner in which the invention was made.

9. Claims 1, 3, 5-6, 9-11, 13, 15-16, 19-23, and 32-33 are rejected under 35 U.S.C. 103(a) as being unpatentable over Parady, U.S. Patent No. 5,933,627 (as applied in the previous Office Action), in view of Kruse and Ryba, "Data Structures And Program Design in C++," 1999 (as applied in the previous Office Action and herein referred to as Kruse). In addition, Hennessy and Patterson, "Computer Architecture - A Quantitative Approach, 2nd Edition," 1996 (as applied in the previous Office Action and herein referred to as Hennessy), is cited as extrinsic evidence for showing different length latency events and the amount of cycles different latency events require.

10. Referring to claim 1, Parady has taught a use of multiple threads including the steps of:
a) providing multiple instruction execution threads as independent processes in a sequential time frame. See Fig.3 and note the implementation of multiple threads. These threads are independent because while one thread is stalled, the system will begin execution of another thread (see the abstract). In addition, each of the threads has its own dedicated resources, such as the registers shown in Fig.3.

b) queuing the multiple execution threads. See Fig.3 and note that each of the threads is queued in instruction buffers 102-108.

c) executing a first thread in a queue. See Fig.3 and note that one of the four threads will initially be executed when its instructions are sent to the dispatch unit.

d) transferring control of the execution to the next thread in the queue upon the occurrence of an event that causes execution of the first thread to stall. See the abstract.

e) returning control to the first thread when the event is completed, if the stall is due to a short latency event. See the abstract. It should be noted that "short" is a relative word, i.e., a "short"

Art Unit: 2183

event may be a “long” event since applicant has not defined in the claims what “long” is defined as. Furthermore, applicant’s “short latency event” is not defined as being 25 cycles or less as applicant argues on page 18 of the appeal brief. Instead, page 4, lines 8-12, of applicant’s specification state “If the latency event is programmed to be short, e.g., 25 machine cycles or less...”. Using an example of 25 machine cycles does not exclude the short latency event from requiring some other amount of cycles larger than 25. Consequently, Parady reads on applicant’s claim. Furthermore, as will be seen later, in the rejection of claim 32, Parady’s “long” latency event actually meets the definition of applicant’s “short” latency event. It should be noted that control will eventually be transferred back to the first thread since it must eventually complete. See column 5, lines 13-14. And, the event must be completed before control may be returned to the thread because it can’t proceed until the event is finished (i.e., it’s data is available to proceed).

f) Parady has not explicitly taught retaining full control if the stall is due to a long latency event. However, memory hierarchies are well known and accepted in the art. A memory hierarchy includes, from fastest to slowest, registers, cache(s), main memory, and hard disk. If data is to be loaded into a register, for instance, an L1 cache would be checked. If the data is not found there, the L2 cache would be checked. If the data is not found there (which is taught by Hennessy), main memory would be checked (this corresponds to the short latency event). However, if main memory does not include the data, then the hard disk must be checked. The examiner asserts that this would be a long latency stall (when the processor must make an I/O request to disk). For known reasons (it provides a large quantity of storage), it would’ve been

obvious to one of ordinary skill in the art to modify Parady to include a hard disk coupled to the processor. Therefore:

1) if a long latency stall occurs for thread A, for instance, then an L2 cache miss must've occurred, and consequently, thread A was switched out and thread B was switched in. Clearly, while this long event takes place, thread B will retain full control. **NOTE: Applicant has not claimed that the full control is retained after the long latency event has completed.**

2) Also, as an alternate interpretation, if a long latency stall occurs for thread A, for instance, then an L2 cache miss must've occurred, and consequently, thread A was switched out and thread B was switched in. When thread A is returned to, it will retain full control if the stall was due to a long latency event. **NOTE: Applicant has not claimed that the full control is retained by the next thread.**

g) Parady has not explicitly taught that the multiple threads have overlapping access to the accessible data available in said tree search structure. However, Kruse has taught that search trees are quick and efficient for inserting, deleting, and searching for data. See section 10.2 on pages 444-445. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Parady such that the threads have access to data available in a tree search structure. Again, one would be motivated to have such a structure because they allow for fast searching, inserting, and deleting of data.

11. Referring to claim 3, Parady in view of Kruse has taught a use as described in claim 1. Parady has further taught that a processor instruction is encoded to select a short latency event. Recall that Parady has taught thread-switching when a cache-miss occurs. See the abstract. It is inherent that such a short latency event (i.e. a cache miss) would result from trying to load data

Art Unit: 2183

that is not in the cache, for example. It is the load instruction's encoding that causes the processor to access the cache and select a short latency event upon a cache miss.

12. Referring to claim 5, Parady in view of Kruse has taught a use as described in claim 1. Parady has further taught that a processor instruction is encoded to select a long latency event. Recall that Parady has taught thread-switching when a disk access occurs (this is inherent because if for a disk access to occur, an L2 cache miss must've occurred, and threads are switched upon L2 cache misses). See the abstract. It is inherent that such a long latency event (i.e. a disk access) would result from trying to load data that is not in the cache or main memory. It is the load instruction's encoding that causes the processor to select a long latency event upon a disk access.

13. Referring to claim 6, Parady in view of Kruse has taught a use as described in claim 1. Parady has further taught queuing the threads to provide rapid distribution of access to shared memory. See Fig.1 and Fig.5 and note that at the very least, the threads share data cache 56/166.

14. Referring to claim 9, Parady in view of Kruse has a taught a use as described in claim 1. Parady has further taught that the threads are used with zero overhead to switch execution from one thread to the next. Parady makes no mention of switch overhead. Consequently, Parady has taught that the switching requires no overhead.

15. Referring to claim 10, Parady in view of Kruse has a taught a use as described in claim 9. Parady has further taught that each thread is given access to general-purpose registers and local data storage to enable switching with zero overhead. See Fig.3 and note that each thread has access to its own set of integer and register files 48 and 50 and PA registers 110.

16. Referring to claim 11, Parady has taught a processor that uses multiple threads to access data, including:

a) a CPU configured with multiple instruction execution threads as independent processes in a sequential time frame. See Fig.3 and note the implementation of multiple threads. These threads are independent because while one thread is stalled, the system will begin execution of another thread (see the abstract). In addition, each of the threads has its own dedicated resources, such as the registers shown in Fig.3.

b) a thread execution control for:

b1) queuing the multiple execution threads to have overlapping access to the accessible data to be accessed. See Fig.3 and note that each of the threads is queued in instruction buffers 102-108. Also, Parady has taught that the threads have overlapping access to the data to be accessed. "Overlapping" is defined by dictionary.com to mean "related by having something in common with or coinciding with." Therefore, looking at Fig.1 and Fig.3, components 48 and 50, regardless of which thread is executing, data may be accessed from register file 48 (in the case of an integer access) and register file 50 (in case of a float access). That is, the common feature to the integer data accesses is that each one will occur in register file 48 (similarly, floats will be accessed from file 50).

b2) executing a first thread in a queue. See Fig.3 and note that one of the four threads will initially be executed when its instructions are sent to the dispatch unit.

b3) transferring control of the execution to the next thread in the queue upon the occurrence of an event that causes execution of the first thread to stall, said thread execution control including control logic responsive to the type of event causing the

execution to stall that transfers full control of the execution to the next thread when execution of the first thread stalls due to a long latency event with execution continuing on the next thread even after the long latency event is completed. See the abstract, and note that if a next thread takes control after a first thread encounters a long latency event, then the next thread will retain control until it encounters a long latency event (regardless of when the first thread's event completes.

NOTE: Applicant's use of the word "or" before section 3(b) of the claim indicates that the transfer control logic is for performing the action specified in part 3(a) **or** for performing the action specified in part 3(b). The prior art, therefore, only needs to teach one of 3(a) and 3(b) to read on the claim.

c) Parady has not taught that the data is available in a tree search structure. However, Kruse has taught that search trees are quick and efficient for inserting, deleting, and searching for data. See section 10.2 on pages 444-445. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Parady such that the threads have access to data available in a tree search structure. Again, one would be motivated to have such a structure because they allow for fast searching, inserting, and deleting of data.

17. Referring to claim 13, Parady in view of Kruse has taught a system as described in claim 11. Parady has further taught that a processor instruction is encoded to select a short latency event. See Fig.4, and note that instructions have register fields, which indicates that instructions will access register files. An access to a register file, while taking minute time, still requires some amount of time. This time is referred to as a short latency event.

18. Referring to claim 15, Parady in view of Kruse has taught a system as described in claim 11. Parady has further taught that a processor instruction is encoded to select a long latency event. Recall that Parady has taught thread-switching when a level-2 cache-miss occurs. See the abstract. It is inherent that such a long latency event (i.e. an L2 cache miss) would result from trying to load data that is not in the cache, for example. It is the load instruction's encoding that causes the processor to access the cache and select a long latency event upon a cache miss.

19. Referring to claim 16, Parady in view of Kruse has taught a system as described in claim 11. Parady has further taught means to queue the threads to provide rapid distribution of access to shared memory. See Fig.1 and Fig.5 and note that at the very least, the threads share data cache 56/166.

20. Referring to claim 19, Parady in view of Kruse has a taught a system as described in claim 11. Parady has further taught that the processor uses zero overhead to switch execution from one thread to the next. Parady makes no mention of switch overhead. Consequently, Parady has taught that the switching requires no overhead.

21. Referring to claim 20, Parady in view of Kruse has a taught a system as described in claim 19. Parady has further taught that each thread is given access to an array of general-purpose registers and local data storage to enable switching with zero overhead. See Fig.3 and note that each thread has access to its own set of integer and register files 48 and 50 and PA registers 110.

22. Referring to claim 21, Parady in view of Kruse has a taught a system as described in claim 20. Parady has further taught that the local data storage is made available to the processor by providing one address bit under the control of the thread execution control logic and by

providing the remaining address bits under the control of the processor. See Fig.3 and Fig.4. It should be noted that in order to use the appropriate register file, a thread number must be provided via field 118 shown in Fig.4. These 2 bits (which include one address bit) will select which register file will be used, for instance. In addition, it is inherent that the processor will address the appropriate registers during execution, i.e., a register address must be specified.

23. Referring to claim 22, Parady in view of Kruse has taught a system as described in claim 20. Parady has further taught that the processor is capable of simultaneously addressing multiple register arrays, and the thread execution control logic includes a selector to select which array will be delivered to the processor for a given thread. See Fig.7 and note that although multiple shadow files may be addressed simultaneously, only one will be enabled, causing that enabled data to be sent to the processor via selector 192.

24. Referring to claim 23, Parady in view of Kruse has taught a system as described in claim 20. Parady has further taught that the local data storage is fully addressable by the processor and the thread execution control has no address control over the local data stored or the register arrays. See Fig.3, component 56, and note that the data cache inherently has locations that are addressable by the processor. Also, a plain data cache access has nothing to do with what thread is being executed. For instance, if data needs to be loaded from the cache, the load address is all that is needed to access the cache, not the thread number. Parady has further taught that an index register is contained within the register array. See Fig.3, component 110. Each of these registers is an index register in that they hold indexes into instruction memory from which thread instructions will be fetched.

Art Unit: 2183

25. Referring to claim 32, Parady in view of Kruse has taught a method as described in claim

1. Parady in view of Kruse has not explicitly taught that a machine cycle is approximately between 5 and 7.5 nanoseconds. However, this range corresponds to processor speeds of 133-200 MHz ($7.5^{-1} \cdot 10^9$ to $5^{-1} \cdot 10^9$). Official Notice is taken that such processor speeds are well known in the art and the speed of the processor is a design choice. In addition, even if the processor of Parady in view of Kruse were faster or slower than 133-200 MHz, a change in size/range is generally not given patentable weight or would have been an obvious improvement (In re Rose, 105 USPQ 237 (CCPA 1955)). Therefore, it would have been obvious to one of ordinary skill in the art to have Parady in view of Kruse's processor be in the range of 133-200 MHz. Furthermore, recall that Parady has taught switching threads due to a cache miss. See the abstract. Even though a cache miss is referred to as "long" latency event by Parady, "long" is a relative word, and this event actually corresponds to a short latency event as defined by applicant. Looking at Hennessy, page 40, Figure 1.15, a cache miss requires an access to main memory, which requires 100ns. Using the 5 to 7.5 machine cycle speeds above, a main memory access would require 13.3 to 20 machine cycles, which is below applicant's 25 or less machine cycle threshold.

26. Referring to claim 33, Parady in view of Kruse has taught a method as described in claim

1. Parady in view of Kruse has not explicitly taught that a machine cycle is approximately between 5 and 7 nanoseconds. However, this range corresponds to processor speeds of 143-200 MHz ($7^{-1} \cdot 10^9$ to $5^{-1} \cdot 10^9$). Official Notice is taken that such processor speeds are well known in the art and the speed of the processor is a design choice. In addition, even if the processor of Parady in view of Kruse were faster or slower than 143-200 MHz, a change in size/range is

Art Unit: 2183

generally not given patentable weight or would have been an obvious improvement (In re Rose, 105 USPQ 237 (CCPA 1955)). Therefore, it would have been obvious to one of ordinary skill in the art to have Parady in view of Kruse's processor be in the range of 143-200 MHz.

Furthermore, recall that Parady has taught switching threads due to a disk access. This event corresponds to a long latency event as defined by applicant. Looking at Hennessy, page 40, Figure 1.15, a disk access requires 5 ms (5,000,000 ns). Using the 5 to 7 machine cycle speeds above, a disk access would require over 700,000 machine cycles, which is well over applicant's 25 machine cycle threshold.

27. Claims 8 and 18 are rejected under 35 U.S.C. 103(a) as being unpatentable over Parady in view of Kruse, as applied above, and further in view of Flynn et al., U.S. Patent No. 6,052,708 (as applied in the previous Office Action and herein referred to as Flynn).

28. Referring to claim 8, Parady in view of Kruse has taught a use as described in claim 1. Parady has further taught a separate instruction buffer for each execution thread. See Fig.3. Parady in view of Kruse has not explicitly taught collecting instructions in a prefetch buffer for its execution thread when the thread is idle and when the instruction bandwidth is not being fully utilized. However, Flynn has taught such a concept. See column 4, lines 43-53. Note that when a thread is inactive, a thread's instructions may be fetched so that when the thread is made active, the instruction will be ready for dispatch immediately, thereby increasing efficiency. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Parady in view of Kruse to prefetch instructions for an inactive thread into its prefetch buffer.

29. Referring to claim 18, Parady in view of Kruse has taught a processing system as described in claim 11. Parady has further taught a separate instruction buffer for each execution thread. See Fig.3. Parady in view has not explicitly taught means for collecting instructions in a prefetch buffer for its execution thread when the thread is idle and when the instruction bandwidth is not being fully utilized. However, Flynn has taught such a concept. See column 4, lines 43-53. Note that when a thread is inactive, a thread's instructions may be fetched so that when the thread is made active, the instruction will be ready for dispatch immediately, thereby increasing efficiency. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Parady to prefetch instructions for an inactive thread into its prefetch buffer.

30. Claims 1, 3, 5-7, 9-11, 13, 15-17, 19-23, and 32 are rejected under 35 U.S.C. 103(a) as being unpatentable over Joy, U.S. Patent No. 6,341,347 (as applied in the previous Office Action and herein referred to as Joy), in view of Kruse, as applied above. In addition, Hennessy is again cited as extrinsic evidence for showing different length latency events and the amount of cycles different latency events require.

31. Referring to claim 1, Joy has taught a use of multiple threads including the steps of:

- a) providing multiple instruction execution threads as independent processes in a sequential time frame. See Fig.2B and column 6, lines 59-66.
- b) Joy has further taught that one of multiple threads may be selected for execution based on priority. See column 4, lines 20-22. Joy has not explicitly taught queuing the multiple execution threads. However, Kruse has taught the well-known concept of a priority queue. A priority

queue, according to Kruse on pages 369-370, is specifically useful in time-sharing computer systems, which have a number of tasks (threads), each of the tasks having a corresponding priority. The priority queue allows for the finding and removal of the entry having the highest priority. Consequently, since Joy has taught one of multiple tasks are selected based on priority, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Joy to include a priority queue to hold these tasks because Kruse has taught that the priority queue is a structure which is able to provide such functionality.

c) executing a first thread in the queue. See column 4, lines 20-22. The highest-priority thread that is awaiting processing will be selected for execution.

d) transferring control of the execution to the next thread in the queue upon the occurrence of an event that causes execution of the first thread to stall. See Fig.2B and column 6, lines 59-66.

e) returning control to the first thread when the event is completed, if the stall is due to a short latency event. See column 2, lines 2-4, column 4, lines 20-22, column 6, lines 59-66, and Fig.2B. Note that on an L1 cache miss, execution control is transferred to the next highest priority thread that is waiting idle. Further note that the L1 cache miss can be considered a short latency event since the applicant has failed to define what constitutes a short latency event.

Furthermore, as will be seen later, in the rejection of claim 32, Joy's L1 cache miss actually meets the definition of applicant's "short" latency event. Finally, from Fig.2B, it can be seen that control is returned to the original thread after it is done stalling.

f) retaining full control if the stall is due to a long latency event. See column 2, lines 21-26, and note that a thread switch may occur on an L2 cache miss. This is a long latency event, even according to applicant's definition, as will be shown later. Therefore:

1) if a long latency stall occurs for thread A, for instance, then an L2 cache miss must've occurred, and consequently, thread A was switched out and thread B was switched in. Clearly, while this long event takes place, thread B will retain full control. **NOTE: Applicant has not claimed that the full control is retained after the long latency event has completed.**

2) Also, as an alternate interpretation, if a long latency stall occurs for thread A, for instance, then an L2 cache miss must've occurred, and consequently, thread A was switched out and thread B was switched in. When thread A is returned to, it will retain full control if the stall was due to a long latency event (i.e., it will retain full control regardless the length of the stall). **NOTE: Applicant has not claimed that the full control is retained by the next thread.**

g) Joy has not explicitly taught that the multiple threads have overlapping access to the accessible data available in said tree search structure. However, Kruse has taught that search trees are quick and efficient for inserting, deleting, and searching for data. See section 10.2 on pages 444-445. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Joy such that the threads have access to data available in a tree search structure. Again, one would be motivated to have such a structure because they allow for fast searching, inserting, and deleting of data.

32. Referring to claim 3, Joy in view of Kruse has taught a use of multiple threads as described in claim 1. Joy has further taught that a processor instruction is encoded to select a short latency event. Recall that Joy has taught thread-switching when an L1 cache-miss occurs. See column 2, lines 2-30. It is inherent that such a short latency event would result from trying to load data that is not in the L1 cache. Therefore, load instructions that are encoded in such a

way to cause the processor to access, and miss the L1 cache, would actually select a short latency event.

33. Referring to claim 5, Joy in view of Kruse has taught a use of multiple threads as described in claim 1. Joy has further taught that a processor instruction is encoded to select a long latency event. Recall that Joy has taught thread-switching when an L2 cache-miss occurs. See column 2, lines 21-26. It is inherent that such a long latency event (i.e. a slower main memory access) would result from trying to load data that is not in the L2 cache. Therefore, load instructions that are encoded in such a way to cause the processor to access, and miss the L2 cache, would actually select a long latency event.

34. Referring to claim 6, Joy in view of Kruse has taught a use of multiple threads as described in claim 1. Joy has further taught queuing the threads to provide rapid distribution of access to shared memory. See Fig.3, components 330 and 320, and column 9, lines 1-5, and lines 36-40.

35. Referring to claim 7, Joy in view of Kruse has taught a use of multiple threads as described in claim 1. Joy has further taught that the threads have overlapping access to shared remote storage via a pipelined coprocessor by operating within different phases of a pipeline of the coprocessor. See Fig.9 and column 19, lines 6-58. Note that the second processor (co-processor) is pipelined. Further note that the co-processor threads have overlapping access to main memory via MIU 928 as well as to all caches.

36. Referring to claim 9, Joy in view of Kruse has taught a use of multiple threads as described in claim 1. Joy has further taught that the threads are used with zero overhead to switch execution from one thread to the next. See column 6, lines 34-36, and note that the

Art Unit: 2183

replication of registers for each thread would result in zero switching overhead, as discussed in column 4, lines 12-16. In addition, Joy makes no mention of switch overhead. Consequently, Joy has taught that the switching requires no overhead.

37. Referring to claim 10, Joy in view of Kruse has taught a use of multiple threads as described in claim 9. Joy has further taught that each thread is given access to general-purpose registers and local data storage to enable switching with zero overhead. See column 6, lines 34-36, column 4, lines 12-16, and column 8, lines 59-67. Note that the registers are replicated for each thread and each thread is also given its own load and store buffers to hold any pending data coming from or going to memory. Furthermore, from Fig.7A, it can be seen that the data cache is divided into separate parts for each thread.

38. Referring to claim 11, Joy has taught a processor that uses multiple threads to access data, including:

- a) a CPU configured with multiple instruction execution threads as independent processes in a sequential time frame. See Fig.2B and column 6, lines 59-66.
- b) Joy has not taught a thread execution control for selecting one of multiple threads for execution based on priority. See column 4, lines 20-22. Joy has not explicitly taught queuing the multiple execution threads. However, Kruse has taught the well-known concept of a priority queue. A priority queue, according to Kruse on pages 369-370, is specifically useful in time-sharing computer systems, which have a number of tasks (threads), each of the tasks having a corresponding priority. The priority queue allows for the finding and removal of the entry having the highest priority. Consequently, since Joy has taught one of multiple tasks are selected based on priority, it would have been obvious to one of ordinary skill in the art at the time of the

Art Unit: 2183

invention to modify Joy to include a priority queue to hold these tasks because Kruse has taught that the priority queue is a structure which is able to provide such functionality. In addition, Joy has taught that the threads have overlapping access to the data to be accessed. See Fig.1B, for instance, and note that when threads are switched, they have overlapping access to data.

“Overlapping” is defined by dictionary.com to mean “related by having something in common with or coinciding with.” Therefore, looking at Fig.8, regardless of which thread is executing, data may be accessed from cache 810. That is, the common feature to the data access is that they occur in cache 810.

c) Joy in view of Kruse has taught executing a first thread in the queue. See Joy, column 4, lines 20-22. The highest-priority thread that is awaiting processing will be selected for execution.

d) Joy in view of Kruse has taught transferring control of the execution to the next thread in the queue upon the occurrence of an event that causes execution of the first thread to stall, said thread execution control including control logic responsive to the type of event causing the execution to stall that transfers full control of the execution to the next thread when execution of the first thread stalls due to a long latency event with execution continuing on the next thread even after the long latency event is completed. See column 2, lines 2-26, column 4, lines 20-22, column 6, lines 59-66, and Fig.2B. Note that on a cache miss, execution control is transferred to the next highest priority thread that is waiting idle. If a next thread takes control after a first thread encounters a long latency event (cache miss, for instance), then the next thread will retain control until it encounters a long latency event (regardless of when the first thread’s event completes).

NOTE: Applicant's use of the word "or" before section 3(b) of the claim indicates that the transfer control logic is for performing the action specified in part 3(a) **or** for performing the action specified in part 3(b). The prior art, therefore, only needs to teach one of 3(a) and 3(b) to read on the claim.

e) Joy has not taught that the data is available in a tree search structure. However, Kruse has taught that search trees are quick and efficient for inserting, deleting, and searching for data. See section 10.2 on pages 444-445. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Joy such that the threads have access to data available in a tree search structure. Again, one would be motivated to have such a structure because they allow for fast searching, inserting, and deleting of data.

39. Referring to claim 13, Joy in view of Kruse has taught a processing system as described in claim 11. Joy has further taught that a processor instruction is encoded to select a short latency event. See Fig.13, and note the existence of a register file. In a system that has registers, instructions will access registers. An access to a register file, while taking minute time, still requires some amount of time. This time is referred to as a short latency event.

40. Referring to claim 15, Joy in view of Kruse has taught a processing system as described in claim 11. Joy has further taught that a processor instruction is encoded to select a long latency event. Recall that Joy has taught thread-switching when a cache-miss occurs. See column 2, lines 2-30. It is inherent that such a long latency event (i.e. a cache miss and ultimately a slower memory access) would result from trying to load data that is not in the cache, for example. Therefore, load instructions which are encoded in such a way that the processor knows to access cache, would actually select a long latency event upon a cache miss.

Art Unit: 2183

41. Referring to claim 16, Joy in view of Kruse has taught a processing system as described in claim 11. Joy has further taught means to queue the threads to provide rapid distribution of access to shared memory. See Fig.3, components 330 and 320, and column 9, lines 1-5, and lines 36-40.

42. Referring to claim 17, Joy in view of Kruse has taught a processing system as described in claim 16. Joy has further taught that the threads have overlapping access to shared remote storage via a pipelined coprocessor by operating within different phases of a pipeline of the coprocessor. See Fig.9 and column 19, lines 6-58. Note that the second processor (co-processor) is pipelined. Further note that the co-processor threads have overlapping access to main memory via MIU 928 as well as to all caches.

43. Referring to claim 19, Joy in view of Kruse has taught a use of multiple threads as described in claim 11. Joy has further taught that the processor uses zero overhead to switch execution from one thread to the next. See column 6, lines 34-36, and note that the replication of registers for each thread would result in zero switching overhead, as discussed in column 4, lines 12-16. In addition, Joy makes no mention of switch overhead. Consequently, Joy has taught that the switching requires no overhead.

44. Referring to claim 20, Joy in view of Kruse has taught a processing system as described in claim 19. Joy has further taught that each thread is given access to an array of general-purpose registers and local data storage to enable switching with zero overhead. See column 6, lines 34-36, column 4, lines 12-16, and column 8, lines 59-67. Note that the registers are replicated for each thread and each thread is also given its own load and store buffers to hold any

Art Unit: 2183

pending data coming from or going to memory. Furthermore, from Fig.7A, it can be seen that the data cache is divided into separate parts for each thread.

45. Referring to claim 21, Joy in view of Kruse has taught a processing system as described in claim 20. Joy has further taught that the local data storage is made available to the processor by providing one address bit under the control of the thread execution control logic and by providing the remaining address bits under the control of the processor. Regarding the data storage (as shown in Fig.8), Joy has taught that the cache is segregated into a first thread's portion and a second thread's portion. Fig.8 also shows the addressing format for accessing the data cache. Note that index field 812 (split into 823 and 824) includes one bit specified by the thread ID (TID), which is an identification tag unique to each thread. See column 3, lines 52-55. This thread ID, when implemented as part of the index field, provides addresses that are only accessible by the thread associated with the corresponding thread ID. The remaining bits are part of a virtual address, which is supplied by the processor's load/store units. See column 8, lines 59-67. In addition, recall that there are separate registers for each thread (see column 6, lines 34-36). In Fig.13 and column 27, lines 32-37, Joy has disclosed that each plane (register window) 1310 represents a separate group of registers within a 3-dimensional register file. There is a one-to-one mapping of register windows to threads, and consequently, thread switching also results in register window switching so that each thread has its own set of registers. Selecting a window is performed by changing the window pointer. As described in a similar fashion above, it would have been obvious to use the thread ID or a subset of the thread ID to specify which register set will be active at a given time.

46. Referring to claim 22, Joy in view of Kruse has taught a processing system as described in claim 20. Joy has further taught that the processor is capable of simultaneously addressing multiple register arrays, and the thread execution control logic includes a selector to select which array will be delivered to the processor for a given thread. See Fig. 13. Recall from the rejection of claim 21, that each plane 1310 represents an array of registers (register file). From Fig. 13, it can be seen that the register index is decoded such that register N is selected from among each of the arrays. However, the current window pointer 1312 is decoded such that register N from the third window will be selected. Therefore, the current window pointer acts as a selector in that it selects a register window for use by the thread.

47. Referring to claim 23, Joy in view of Kruse has taught a processing system as described in claim 20. Joy has further taught that the local data storage is fully addressable by the processor, an index register is contained within the register array, and the thread execution control has no address control over the local data storage or the register arrays. See Fig. 8 and note the 64-bit indexing format, which, as known in the art, could be stored in one of the 64-bit registers (for indirect addressing purposes) in register file 1300 (Fig. 13). One embodiment, as described in the rejection of claim 21, has part of the index field including a thread ID, which is provided based on the currently executing thread. However, according to column 18, lines 21-38, this thread ID tagging can be disabled in cases where native threads and lightweight processes are sharing the same virtual address space while being executed. This allows for a non-segregated cache, which would eliminate the under-utilization of resources resulting from certain threads.

Art Unit: 2183

48. Referring to claim 32, Joy in view of Kruse has taught a method as described in claim 1. Joy in view of Kruse has not explicitly taught that a machine cycle is approximately between 5 and 7.5 nanoseconds. However, this range corresponds to processor speeds of 133-200 MHz ($7.5 \cdot 10^9$ to $5 \cdot 10^9$). Official Notice is taken that such processor speeds are well known in the art and the speed of the processor is a design choice. In addition, even if the processor of Joy in view of Kruse were faster or slower than 133-200 MHz, a change in size/range is generally not given patentable weight or would have been an obvious improvement (In re Rose, 105 USPQ 237 (CCPA 1955)). Therefore, it would have been obvious to one of ordinary skill in the art to have Joy in view of Kruse's processor be in the range of 133-200 MHz. Furthermore, recall that Joy has taught switching threads due to an L1 cache miss (which results in an access to an L2 cache). See column 2, lines 2-4, and lines 21-25. Looking at Hennessy, page 40, Figure 1.15, an access to an L1 cache would require 3-10 ns. A miss at the L1 cache would require another cache access (at the L2 cache), which would cost another 3-10 ns. Consequently, the total access time would be roughly 6-20 ns. Using the 5 to 7.5 machine cycle speeds above, a main memory access would require between 0.8 machine cycles ($6/7.5$) and 4 machine cycles ($20/5$). This range corresponds to a range that is below applicant's 25 or less machine cycle threshold. Therefore, an L1 cache miss qualifies as a short latency event.

49. Claims 8 and 18 are rejected under 35 U.S.C. 103(a) as being unpatentable over Joy in view of Kruse, as applied above, in view of Parady, as applied above, and further in view of Flynn, as applied above.

50. Referring to claim 8, Joy in view of Kruse has taught a use of multiple threads as described in claim 1.

a) Joy in view of Kruse has not explicitly taught providing a separate instruction buffer for each execution thread. However, Parady has taught the concept of providing separate buffers for each execution thread. See column 5, lines 6-10. Parady has further disclosed that upon a thread switch, the stream of instructions in one of the instruction buffers will simply pick up where it left off. See column 3, lines 51-56. This would eliminate having to flush a single instruction buffer (if only one buffer were used to store instructions for all of the threads) and refilling it with the correct thread's instructions. Therefore, it would have been obvious to one of ordinary skill in the art at the time of the invention to provide a prefetch buffer for each thread.

b) Joy in view of Kruse in view of Parady has taught providing separate buffers for each execution thread. Joy in view of Kruse in view of Parady has not explicitly taught collecting instructions in a prefetch buffer for its execution thread when the thread is idle and when the instruction bandwidth is not being fully utilized. However, Flynn has taught such a concept. See column 4, lines 43-53. Note that when a thread is inactive, a thread's instructions may be fetched so that when the thread is made active, the instruction will be ready for dispatch immediately, thereby increasing efficiency. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Joy in view of Kruse in view of Parady such that each buffer is a prefetch buffer and instructions for an inactive thread are prefetched into its prefetch buffer.

51. Referring to claim 18, Joy in view of Kruse has taught a processing system as described in claim 11.

Art Unit: 2183

a) Joy in view of Kruse has not explicitly taught providing a separate instruction buffer for each execution thread. However, Parady has taught the concept of providing separate buffers for each execution thread. See column 5, lines 6-10. Parady has further disclosed that upon a thread switch, the stream of instructions in one of the instruction buffers will simply pick up where it left off. See column 3, lines 51-56. This would eliminate having to flush a single instruction buffer (if only one buffer were used to store instructions for all of the threads) and refilling it with the correct thread's instructions. Therefore, it would have been obvious to one of ordinary skill in the art at the time of the invention to provide a prefetch buffer for each thread.

b) Joy in view of Kruse in view of Parady has taught providing separate buffers for each execution thread. Joy in view of Kruse in view of Parady has not explicitly taught collecting instructions in a prefetch buffer for its execution thread when the thread is idle and when the instruction bandwidth is not being fully utilized. However, Flynn has taught such a concept. See column 4, lines 43-53. Note that when a thread is inactive, a thread's instructions may be fetched so that when the thread is made active, the instruction will be ready for dispatch immediately, thereby increasing efficiency. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Joy in view of Kruse in view of Parady such that each buffer is a prefetch buffer and instructions for an inactive thread are prefetched into its prefetch buffer.

52. Claims 27 and 29-31 are rejected under 35 U.S.C. 103(a) as being unpatentable over Cutler et al., U.S. Patent No. 5,752,031 (as applied in the previous Office Action and herein referred to as Cutler), in view of Cota-Robles, U.S. Patent No. 6,658,447 (herein referred to as

Cota), and further in view of Nomura et al., U.S. Patent No. 5,327,526 (herein referred to as Nomura).

53. Referring to claim 27, Cutler has taught a thread execution control useful for the efficient execution of independent threads comprising:

a) a priority FIFO buffer for storing thread numbers (see Fig.4-5 and note the ready queue), said buffer including:

a1) means for loading a thread number into the buffer when a packet is dispensed to the processor. See Fig.4 and column 9, lines 63, to column 10, line 3, and note that a request packet is sent to the processor and a thread is initialized and loaded into the ready queue.

a2) means for unloading a thread number from the buffer when a packet has been enqueued for transmission. Clearly, a thread that is loaded into the queue will be removed when it is set to execute. Also, when a thread is finished, it should be removed from the queue.

a3) Cutler has not taught thread number transfer from highest priority to lowest priority in the buffer when a long latency event occurs. However, Cota has taught that threads that have high execution latencies (for instance, accesses to slower processor resources), should be made lower priority threads. See column 7, lines 48-57. As would be realized by one of ordinary skill in the art, a thread that will require a lot of time to execute (due to slower resource access) will also hold up other threads (even those that may execute quickly). Consequently, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Cutler such that when a long latency event occurs in Cutler, the corresponding thread is assigned lower priority. By doing this, it is ensured that this thread will not hold up higher priority threads.

a4) Cutler has not taught thread outlets of the buffer used to determine priority depending on the length of time a thread has been in the buffer. However, Nomura has taught the general concept of increasing priority of jobs in a queue based on the amount of time spent in the queue. See Fig. 7A-7F and column 6, lines 13-66, and column 7, lines 12-22. As disclosed by Nomura, such a scheme, which is known in the art, allows an initially low priority job to eventually be executed even though initially higher priority jobs may be queued. This essentially prevents the low priority job from never being executed if all if nothing but higher priority jobs are subsequently queued. This scheme also applies to threads in that if there is always a higher priority thread in the queue, a low priority thread would never execute. Nomura's scheme, however, ensures that over time, low priority threads would be executed. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Cutler to include the priority-increasing scheme of Nomura.

b) Cutler has further taught a plurality of thread control state machines, one for each thread. See Fig. 4, and note that each thread's state must be tracked (i.e., it must be tracked whether each thread is in the ready state, standby state, running state, waiting state, etc., so that the next state for the thread may be determined).

c) Cutler in view of Cota and further in view of Nomura has taught an arbiter for determining the thread execution priority among multiple threads based upon signals outputted from the FIFO buffer and the state machines. It is inherent that if one of a plurality of threads is selected, then some component, more specifically, an arbiter, must determine which thread to select. The highest-priority thread that is awaiting processing will be selected for execution. See Fig. 4-5 of Cutler and also recall the discussion of Cota and Nomura. This selection is based on signals

Art Unit: 2183

from the state machines (signals specifying what state the threads are in...the only threads that may be selected for dispatch/execution are those in the ready/standby state) and signals from the FIFO (priorities).

54. Referring to claim 29, Cutler in view of Cota and further in view of Nomura has taught a thread execution control as described in claim 27. Cutler in view of Cota and further in view of Nomura has further taught that the arbiter controls the priority of execution of multiple independent threads based on the Boolean expression recited in claim 29 comprising:

- a) determining whether a request R is active or inactive. See Fig.4 of Cutler and note that only threads in the ready/standby state can make active requests for dispatch/execution. Threads being initialized, or terminated, or those that are already executing, do not need to make requests for execution. Clearly, each thread is associated with a request, but for those that have already had their request satisfied, and are executing, they're requests are inactive (i.e., they're not making requests anymore).
- b) determining the priority of the threads. See Cutler, Fig.4, and note that threads are preempted by higher priority threads. Also recall the discussion of Cota and Nomura above.
- c) matching the request R with the corresponding thread P. Clearly, only requesting (ready) threads may be executed. Of the requesters, only the highest priority will be chosen. See Fig.4-5 of Cutler, and also recall the priority schemes of Cota and Nomura.
- d) granting a request for execution if the request is active and if the corresponding thread P has the highest priority. If a thread is ready to execute and therefore requesting execution, if it is the higher priority thread, then it will be granted execution. See Fig.4-5 of Cutler, and recall Cota and Nomura, which talk about priority of items in a queue.

It should be noted that the formula of claim 29 represents the algorithm set forth in steps (a)-(d) of claim 29, and Joy has taught steps (a)-(d), then Joy has also taught the formula even though it is not explicitly stated.

55. Referring to claim 30, Cutler in view of Cota and further in view of Nomura has taught thread execution control as described in claim 27. Cutler has taught control logic to:

- a) dispatch a packet to a thread. See column 9, lines 63-66, and note that a request packet is received and a thread is initialized to handle the request.
- b) move the thread from the initialize state to a ready state. See Fig.4, and note the path from 40a to 40b.
- c) request execution cycles for the thread. Clearly, by being in the ready state, the threads are requesting execution.
- d) move the thread to the execute state upon grant by the arbiter of an execution cycle. See Fig.4 and note that an arbiter will choose to either preempt a thread standing by for execution or send a thread to an execute stage 40d.
- e) continue to request execution cycles while the thread is queued in the execute state. Again, the thread will continue to request execution cycles until it completes or some other event happens (preemption or lack of resources).
- f) return the thread to the initialize state if there is no latency event, or send the thread to the wait state upon occurrence of a latency event. See Fig.4.

56. Referring to claim 31, Cutler in view of Cota and further in view of Nomura has taught thread execution control as described in claim 27. Cutler has further taught means to detect

occurrence of latency events. See the path from 40d to 40e, and note that a thread must wait for kernel objects. This waiting is a latency event.

Response to Arguments

57. Regarding the “overlapping” argument on page 10, the examiner asserts that his interpretation of “overlapping access” is set forth in the related rejections above.

58. Regarding the “independent processes” argument on page 12, the examiner asserts that “independent processes” does not necessarily mean that the processes are part of separate programs. Instead, a thread may be viewed as independent because they are executed independently. That is, when thread 1 executes, thread 2 is waiting to execute. When thread 2 executes, thread 1 is waiting. The fact that they execute separately makes them independent. And, they are also independent sub-programs, even if they are part of the same overall program.

59. All other arguments have either been addressed in the rejections above or are moot in view of the new grounds of rejection.

Conclusion

60. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure. Applicant is reminded that in amending in response to a rejection of claims, the patentable novelty must be clearly shown in view of the state of the art disclosed by the references cited and the objections made. Applicant must also show how the amendments avoid such references and objections. See 37 CFR § 1.111(c).

Fiske et al., "Thread Prioritization : A Thread Scheduling Mechanism For Multiple-Context Parallel Processors," has taught prioritizing threads in a queue which holds thread numbers.

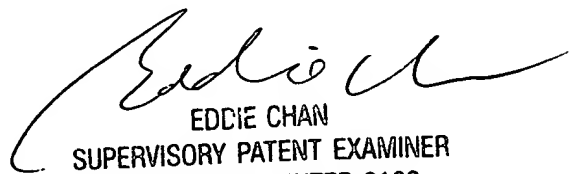
Weiss, "Data Structures & Algorithm Analysis in C++, 2nd Edition," has taught a priority queue where items are loaded into a queue from the rear and retrieved using a delete function, which finds the highest/lowest priority item in the queue.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to David J. Huisman whose telephone number is (571) 272-4168. The examiner can normally be reached on Monday-Friday (8:00-4:30).

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Eddie Chan can be reached on (571) 272-4162. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see <http://pair-direct.uspto.gov>. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

DJH
David J. Huisman
October 10, 2005


EDDIE CHAN
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2100